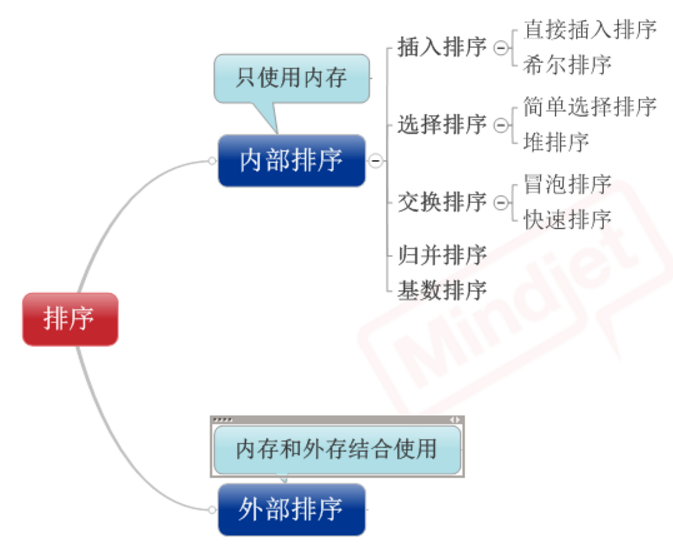
# 常用排序算法

总结各排序算法如下：





## 插入排序

#### 算法描述

插入排序是在一个已经有序的小序列的基础上，一次插入一个元素。当然，刚开始这个有序的小序列只有1个元素，就是第1个元素。比较是从有序的序列末尾开始，也就是想要插入的元素和已经有序的最大者开始比较，如果比它大则直接插入在其后面，否则一直往前找直到找到它该插入的位置。如果碰见一个和插入元素相等的，那么插入元素就要放在相等元素的后面。所以相等元素的前后顺序没有改变，所以排序是稳定的。

具体描述：

1. 从第一个元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中从后向前扫描
3. 如果该元素(已排序)大于新元素，将该元素移到下一位置
4. 重复步骤3，直到找到已排序的元素小于或者等于新元素的位置
5. 将新元素插入到该位置
6. 重复步骤2~5

如果目标是把n个元素的序列升序排列，那么采用插入排序存在最好情况和最坏情况。最好情况就是，序列已经是升序排列了，这样需要进行的比较操作是(n-1)次。最坏情况是，序列是降序排列，那么此时需要进行的比较是n(n-1)/2次。

平均来说插入排序算法时间复杂度为O(n^2)，因此插入排序不适合对于数据量比较大的排序应用。但是如果数据量比较小，例如，量级小于千，那么插入排序还是一个不错的选择。

时间复杂度：最好情况(已经排好序的情况)O(n)，最坏情况(与所要排顺序相反)O(n^2)

#### 考察点

基本在笔试选择填空问时间复杂度时才可能出现。毕竟排序速度比较慢，因此在算法大题中考察的速度比较少。

## 二、二分插入排序

#### 算法简介

二分(折半)插入排序是一种在直接插入排序算法上进行小改动的排序算法。其与直接插入排序算法最大的区别是查找插入位置时使用的是二分查找的方式。在速度上有一定提升。

#### 算法描述和分析

1. 从第一个元素开始，该元素可以认为已经被排序
2. 取出下一个元素，在已经排序的元素序列中二分查找到第一个比它大的数的位置
3. 将新元素插入到该位置后
4. 重复上述步骤
5. 稳定
6. 空间代价：O(1)
7. 时间代价：最佳情况O(nlogn)，最差和平均情况O(n^2)

## 考察点

这个排序算法出现的频率也不高，但毕竟是直接排序算法的一个小改进算法，同时二分查找又是很好的思想，可能会在面试的时候提到。

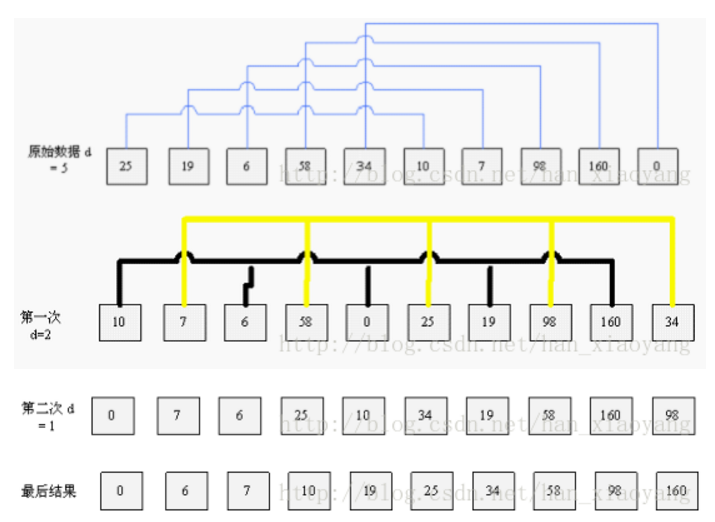
## 三、希尔排序

希尔排序，也称递减增量排序算法，是插入排序的一种高速而稳定的改进版本。基本思想是将相距某个“增量”的记录组成一个子序列，对每个子序列进行直接插入排序，然后减小“增量”，再对每个子序列进行直接插入排序，以此循环，直到“增量”为1，然后进行最后直接插入排序。

增量间隔序列函数h = h\*3+1

希尔排序比插入排序快很多的原因：当h值很大时，数据项每一趟排序移动的元素个数少，但移动的距离很长，这是非常高效的；当h值减小时，每一趟排序移动的元素个数增加，但此时的数据项已经接近于他们最终排序后的位置，插入排序可以更有效。

#### 算法图解



最坏情况时间复杂度：O(n^2)

选择的步长和时间复杂度的关系

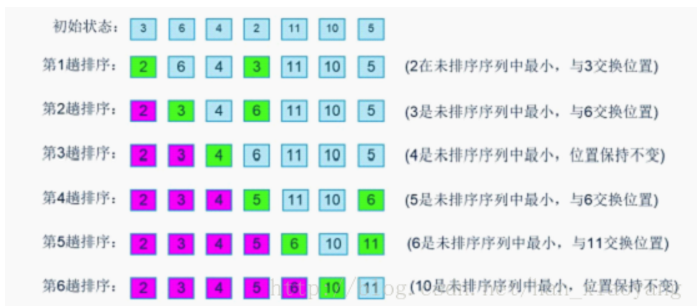
## 四、选择排序

#### 算法简介

首先在未排序序列中找到最小(大)元素，与排序序列的起始位置元素交换，然后，再从剩余未排序元素中继续寻找最小(大)元素，然后与已排序序列元素后面的元素进行交换。以此类推，直到所有元素均排序完成。

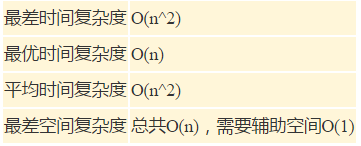


#### 算法图解

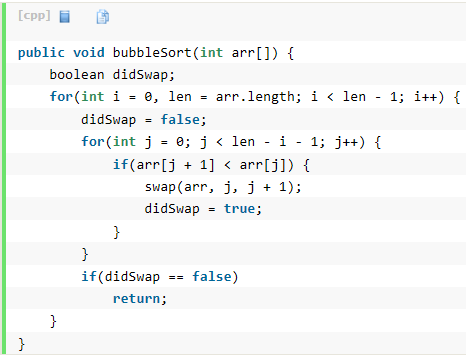


## 五、冒泡排序

#### 时间复杂度



这里要说下为什么最优时间复杂度是O(n)。因为之前的O(n＾2)是一种通用写法，并没考虑最好的情况，即已经排好序。可以在外层遍历的过程当中设置一个标志位为false，如果在循环的过程中没有发生交换，说明已经排好序，那么标志位为false，如果发生了交换那么标志位就为true，这样在后面判断为true时直接退出。



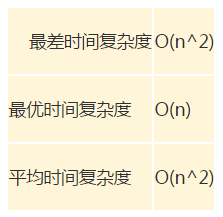
#### 考点

冒泡排序是一个很常见的考点，平均时间空间复杂度，最好最坏情况下的时间复杂度，在不同情况下每一趟的比较次数，以及加标志位减少比较次数等，都是需要注意的地方。

## 六、鸡尾酒排序/双向冒泡排序

第一趟从a到b冒泡，到了b再逆序冒泡到a，第二趟重复第一趟的情况

时间复杂度：



出现频率不高，选择填空会少量出现，并且多以双向冒泡排序出现。

## 七、快速排序

#### 算法简介

通过一趟排序将待排记录分隔成独立的两部分，其中一部分记录的关键字均比另一部分的关键字小，则可分别对这两部分记录继续进行排序，以达到整个队列有序。

#### 步骤

1. 从数列中调出一个元素称为基准，一般是第一个。
2. 从后往前，如果后面的数比基准小，则交换前面与后面的数；然后从前往后，如果前面的数比基准大，则交换前面与后面的数。在这个分区退出之后，该基准就处于数列的中间位置。
3. 递归的把左子列和右子列排序



public static void sort(int[] a,int low,int high){  
 int start = low;  
 int end = high;  
 int key = a[low];  
  
 while(end>start){  
 //从后往前比较  
 while(end>start&&a[end]>=key) //如果没有比关键值小的，比较下一个，直到有比关键值小的交换位置，然后又从前往后比较  
 end--;  
 if(a[end]<=key){  
 int temp = a[end];  
 a[end] = a[start];  
 a[start] = temp;  
 }  
 //从前往后比较  
 while(end>start&&a[start]<=key)//如果没有比关键值大的，比较下一个，直到有比关键值大的交换位置  
 start++;  
 if(a[start]>=key){  
 int temp = a[start];  
 a[start] = a[end];  
 a[end] = temp;  
 }  
 }  
 if(start>low) *sort*(a,low,start-1);//左边序列。第一个索引位置到关键值索引-1  
 if(end<high) *sort*(a,end+1,high);//右边序列。从关键值索引+1到最后一个  
}

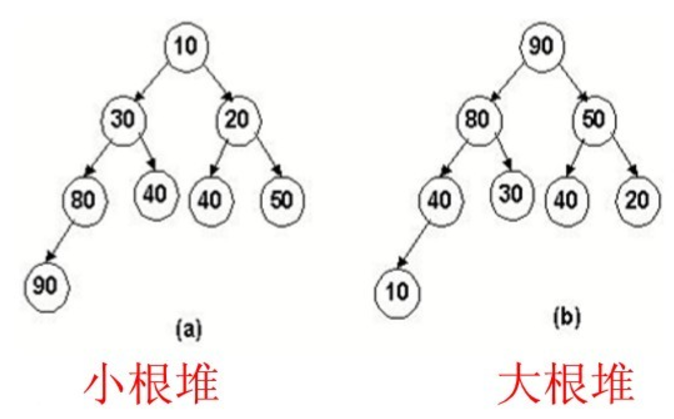
这里要注意判断条件和什么时候退出。

## 八、堆排序

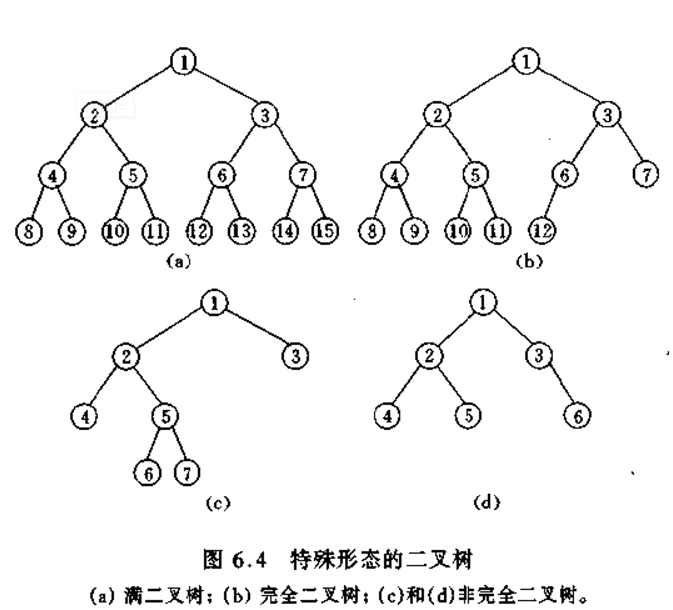
堆排序很容易出现，选择填空问答算法大题都会出现，建堆的过程，堆调整的过程，这些过程的时间复杂度，空间复杂度，以及如何应用在海量数据topK问题中。

**最大堆(大根堆)**：根节点的键值是所有堆节点键值中**最大者**，且每个节点的值都比其孩子的值**大**。

**最小堆(小根堆)**：根节点的键值是所有堆节点键值中**最小者**，且每个节点的值都比其孩子的值**小**。



**完全二叉树**：一个深度为k，节点个数为2＾k – 1的二叉树为满二叉树。即一棵树，深度为k，并且没有空位。然后对满二叉树进行广度优先遍历的顺序进行编号。如果有另外一颗树按照广度优先遍历进行编号，**其所有编号都与满二叉树对应，那么这棵树就是完全二叉树**。



最大堆和最小堆都是一个完全二叉树。

记住一点：设父节点的编号为i(i的范围是[0,n-1])，则左节点的编号为2i+1，右节点的编号为2i+2。

代码如下：

public class heapSort {  
 public static void createMaxHeapforSingleNode (int[] array, int sortLastIndex, int lastIndex) { //对单个根节点，向下递归的创建最大堆  
 if (lastIndex > sortLastIndex) {return;}  
 int i = (lastIndex - 1) / 2;  
 int father = i ;  
 int left = i \* 2 + 1;  
 int right = i \* 2 + 2;  
 if (array[father] < array[left]) {  
 int emp = array[father];  
 array[father] = array[left];  
 array[left] = emp;  
 }  
 if (array[father] < array[right]) {  
 int emp = array[father];  
 array[father] = array[right];  
 array[right] = emp;  
 }  
 *createMaxHeapforSingleNode*(array, sortLastIndex, left \* 2 + 2);  
 *createMaxHeapforSingleNode*(array, sortLastIndex, right \* 2 + 2);  
 }  
  
 */\*\*  
 \* 初始化数组，使其成为最大堆  
 \** ***@param*** *a 数组  
 \** ***@param*** *sortLastIndex 最大堆的最后的下标，不一定是数组的最后的下标  
 \** ***@param*** *lastIndex 数组的最后的下标  
 \*/* public static void initMaxHeap (int[] a, int sortLastIndex, int lastIndex) { //建立最大堆  
 for (int i = (lastIndex - 1) / 2; i >= 0; i--) {  
 *createMaxHeapforSingleNode*(a, sortLastIndex, i \* 2 + 2);  
 }  
 }  
 */\*\*  
 \* 对最大堆进行排序，因为已经知道最大堆的开头元素是最大的，所以如果要升序排列的话，将开头元素与最后元素进行交换，然后从倒第二个元素一直到开头元素重建最大堆，让开头元素与倒第二元  
 \* 素交换，接着从倒第三元素向上重建最大堆，以此循环  
 \** ***@param*** *a 要排序的数组  
 \*/* public static void heapSort(int[] a) {  
 *initMaxHeap*(a, a.length - 1,a.length - 1);  
 for (int i = 0; i < a.length; i++) {  
 int tmp = a[0];  
 a[0] = a[a.length - 1 - i];  
 a[a.length - 1 - i] = tmp;  
 *initMaxHeap*(a, a.length - 1 - i - 1,a.length - 1 - i - 1);  
 }  
 }  
 public static void main(String[] args) {  
 int[] a = {5, 3, 6, 2, 1, 9, 4, 8, 7};  
 *heapSort*(a);  
 System.*out*.print("[ ");  
 for (int i = 0; i < a.length; i++) {  
 System.*out*.print(a[i] + ",");  
 }  
 System.*out*.print("]");  
 }  
}

将判断中的“<”改为“>”就是最小堆。

将数组初始化为最大堆之后，头结点就是最大的元素，让头结点与尾节点交换，然后对倒第二个元素往上重建最大堆，让头结点与倒第二个节点交换，然后对倒第三个元素往上重建最大堆，让头结点与倒第三节点交换，以此类推。

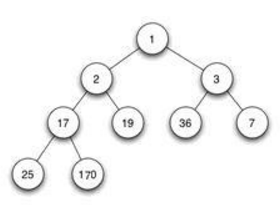
#### 算法简介

堆积是一个近似完全二叉树的结构，并同时满足堆积的性质；即子节点的键值或索引总是小于(或者大于)它的父节点。

我们这里指的堆一般都是指的二叉树，它满足两个特性：

1. 父节点的键值总是大于或等于(小于或等于)任何一个子节点的键值
2. 每个节点的左子树或右子树都是一个二叉堆(都是最大推或最小堆)。

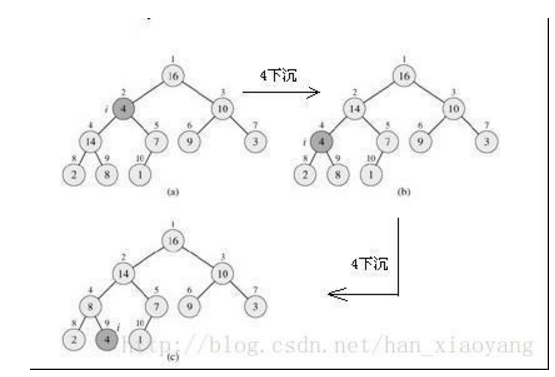
如下为一个最小堆：



堆调整—

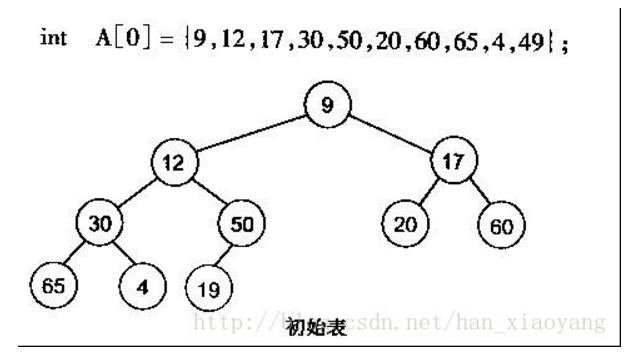
这个为了保证堆的特性而做的一个操作。对某一个节点为根的子树做堆调整，其实就是将该根节点进行“下沉”操作(具体是通过和子节点交换完成的)，一直下沉到合适的位置，使得刚才的子树满足堆的性质。

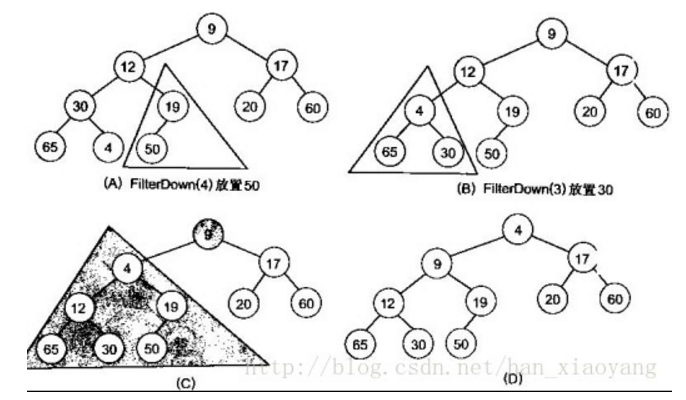
做一次堆调整的时间复杂度为log(n)



如何建堆—

建堆是一个通过不断的堆调整，使得整个二叉树中的数满足堆性质的操作。在数组中的话，我们一般从下标为n/2的数开始做堆调整，一直到下标为0的数(因为下标大于n/2的数都是叶子节点，其子树已经满足堆的性质了)。下图为一个图示：





如何进行堆排序

时间复杂度：



#### 考察点

选择填空算法大题都会出现。建堆的过程，堆调整的过程，时间复杂度、空间复杂度，以及如何应用在海量数据topK中

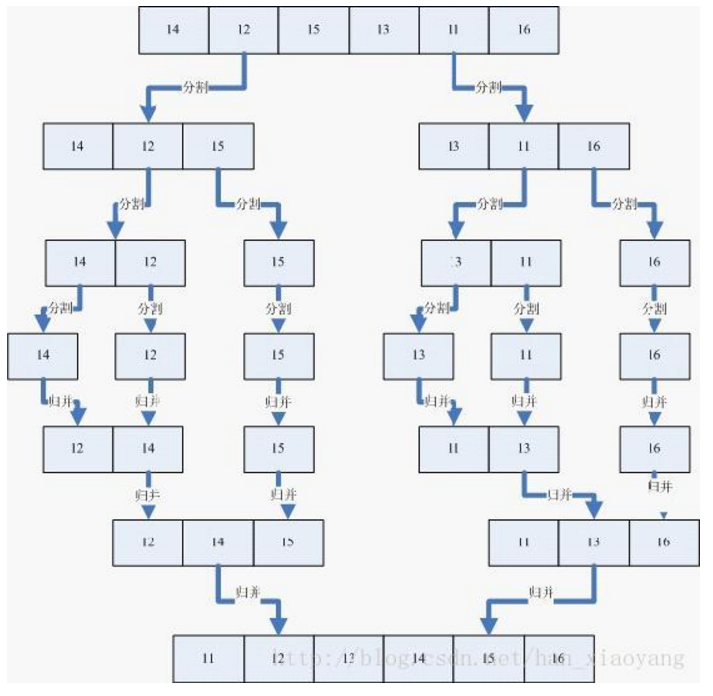
## 九、归并排序

归并排序是建立在归并操作上的一种有效的排序算法。该算法是采用分治法的一个非常典型的应用。归并排序是一种稳定的排序算法。将已有的子序列合并，得到完全有序的序列；即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为2-路归并。

#### 算法描述

1. 把长度为n的输入序列分成两个长度为n/2的子序列。
2. 对这两个子序列分别采用归并排序
3. 将两个排序好的子序列合并成一个最终的排序序列。

图解：



## 十、桶排序

桶排序也叫箱排序，工作原理是将数组分到有限数量的桶子里。每个桶子再个别排序(有可能再使用别的排序算法或是以递归方式继续使用桶排序进行排序)。

桶排序是稳定的，且在大多数情况下是常见排序里最快的一种，比快排还要快，缺点是非常耗空间，基本上是最耗空间的一种排序算法，而且只能在某种情况下使用。

#### 算法描述和分析

桶排序假设数据输入的范围确定，假设为[0,100]，这样便可以建立10个桶，其中每个桶分别存储[0,10),[10,20),[20,30)…….[90,99)之间的元素。把每一个输入数据放到相应的桶里，同时对每一个桶内部进行排序。然后再把桶合并。

桶的个数是max-min+1

桶排序适合元素集合不太大的情况。

## 十一、计数排序

计数排序只能对整数进行排序。计数排序使用一个额外的数组C，其中第i个元素是待排序数组A中值等于i的元素的个数。然后根据数组C来将A中的元素排到正确的位置。

时间复杂度O(n)

## 十二、基数排序

将整数按位数切割成不同的数字，然后按每个位数分别比较。由于整数也可以表达字符串(比如名字和日期)和特定格式的浮点数，所以基数排序也不是只能适用于整数。

#### 算法过程描述如下

1. 将所有待比较数值统一为同样的数位长度，数位较短的数前面补0
2. 从最左位开始，依次进行一次排序
3. 这样从最左位排序一直到最右位排序完成以后，数列就变成一个有序数列

#### 时间复杂度为O(k\*n)

## 位图算法

应用场景举例：

给出10000000个正整数序列，每个数都小于10000000，数据不重复且数据之间不存在关联关系，要求增序输出序列。

限制：内存容量1MB，磁盘空间充足，运行时间至多几分钟------最好线性时间

与计数排序相似，只不过不是用数组而是用二进制数，二进制数的位数由最大的数决定，首先将二进制的所有位全部设为0，然后按照数值大小将对应位上设为1。

比如有数1，2，4，5，6。那么二进制数就是1101110，然后按照顺序将为1的下标输出就行了。